The three random numbers for the seeds will be 724557927, 349891811 and 187982419. This will apply to all experiments so they can be compared to each other throughout.

Experiment 1 - An investigation into the efficiency of the schedulers with a high “meanNumberBursts” with very low mean CPU and IO bursts. This will simulate the conditions that the scheduler will have to contend with when doing many small operations on an average-sized number of processes.

Introduction -
This experiment will be an investigation into the effect on the efficiency of the scheduler under the condition of many small tasks. This will simulate the CPU having to do many very basic tasks, with both many IO and CPU bursts of very short periods.

With this simulation, I hope to demonstrate how the different schedulers behave around having to move many processes on and off of the core. The same amount of processes will be created over the experiment, compared to the control and experiment 1, unlike experiment 1 the processes will arrive at the same frequency as the control.

My hypothesis is that schedulers with the ability to see the shortest job first will potentially suffer more from this test. This is because, if the CPU is being flooded with many small processes with similar small time frames, being able to choose the shortest next burst won't help too much. If the scheduler is wasting time in determining the next shortest process out of 10 processes that all have the same, or very similar, short times then response time and idle time may suffer.
The scheduler would instead benefit by just getting the next process on as quickly as possible, and if the processes all have small burst times, less than that of the time quantum, then it could benefit more. In this regard, round-robin may act similarly to a first come first served scheduler, this is because if it keeps getting processes all of the same priority, all below its time quantum, it will essentially just take the first item the same as the first come first served. This will become more apparent in the first of the round-robin schedulers, because of the fact it has a higher time quantum than its feedback round-robin counterpart, it's more likely to complete the smaller burst tasks within its allotted time. This will be unlike the feedback version, as it is more possible for it to be larger than the initial bust of 8 causing it to be raised in priority and its timer increased.

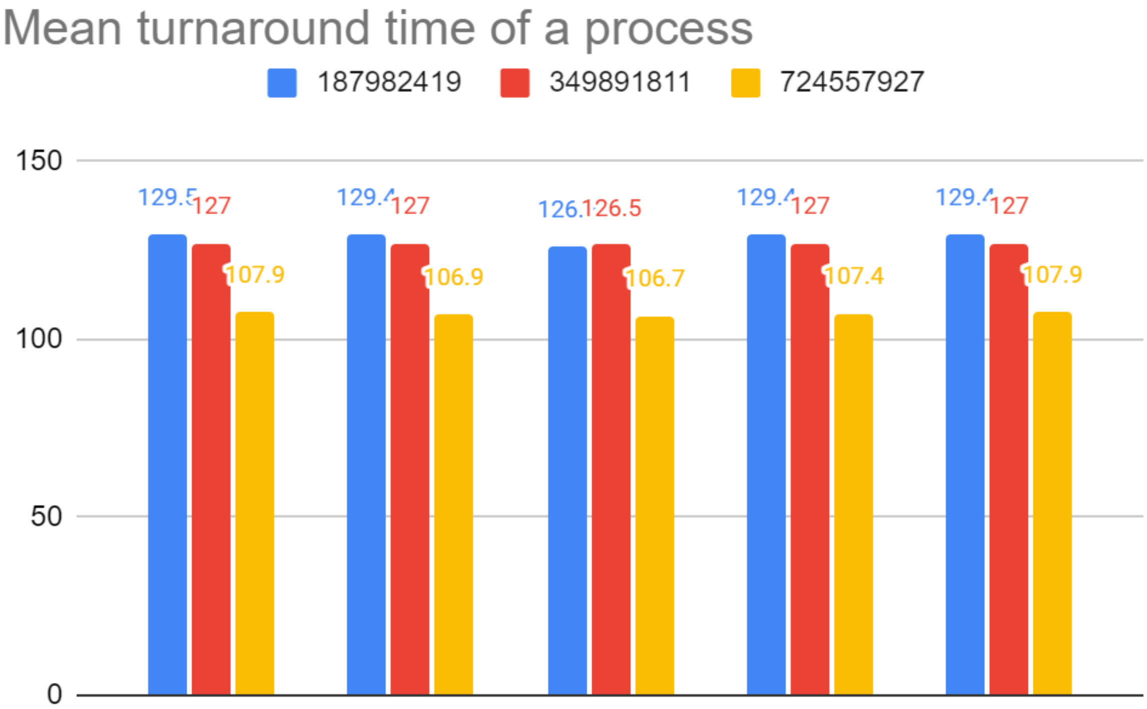
Methodology -

For the first experiment, the change to the input parameters will be similar to the control, however, the mean IO, CPU and the NumberBursts will be essentially swapped. This will be a decrease in the mean IO burst and the mean CPU burst by a factor of 5. Alongside this, there will be an increase in the number of bursts by a factor of 5. The result of this, should in theory across the three seeds, be a more balanced however increased number of bursts per process however with a decreased burst time. There will be no changes to the number of processes, staying at 10 across the experiment the same as the control, however, each process will be longer in the sense of things the scheduler has to do. With this, the static priority and interarrival time will also remain the same as the control meaning that they should arrive at the same time.

This experiment will be run on all five of the schedulers, across all three of the seeds. By doing this, I hope to demonstrate some kind of difference in the turnaround time, waiting time, response time and CPU time of the idle process.

Results -

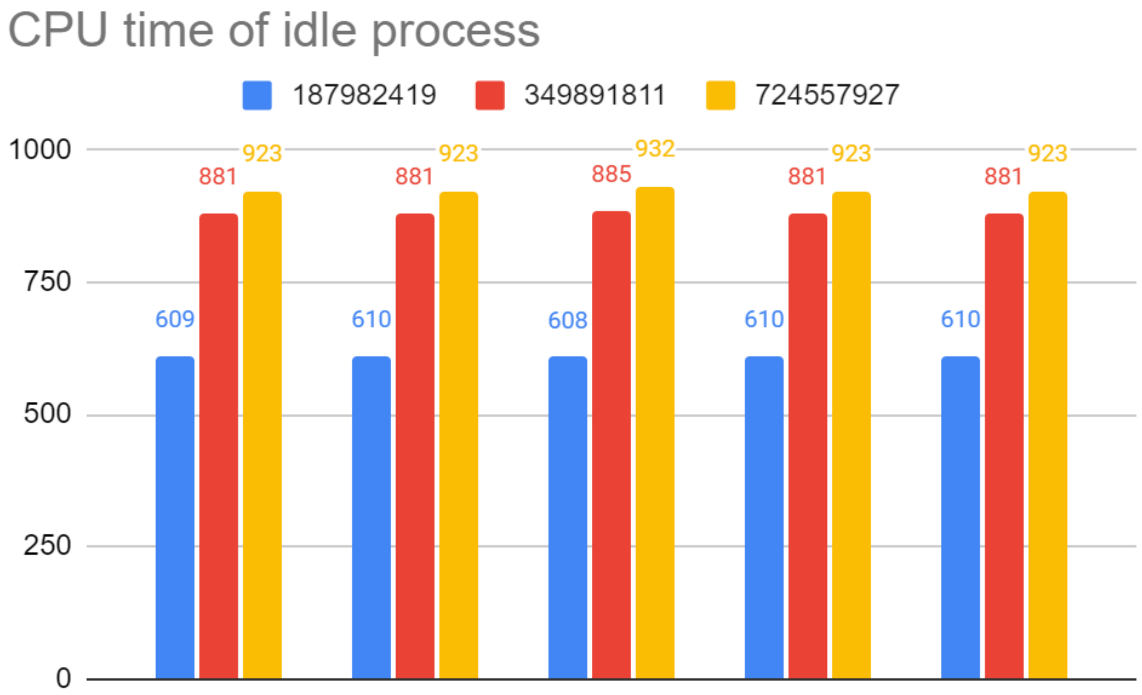
The first analysis will be of the mean turnaround time of a process. This is taken from creating a mean of the processes turnaround time for each of the seeds and on each of the schedulers. This data is represented below in figure 1.



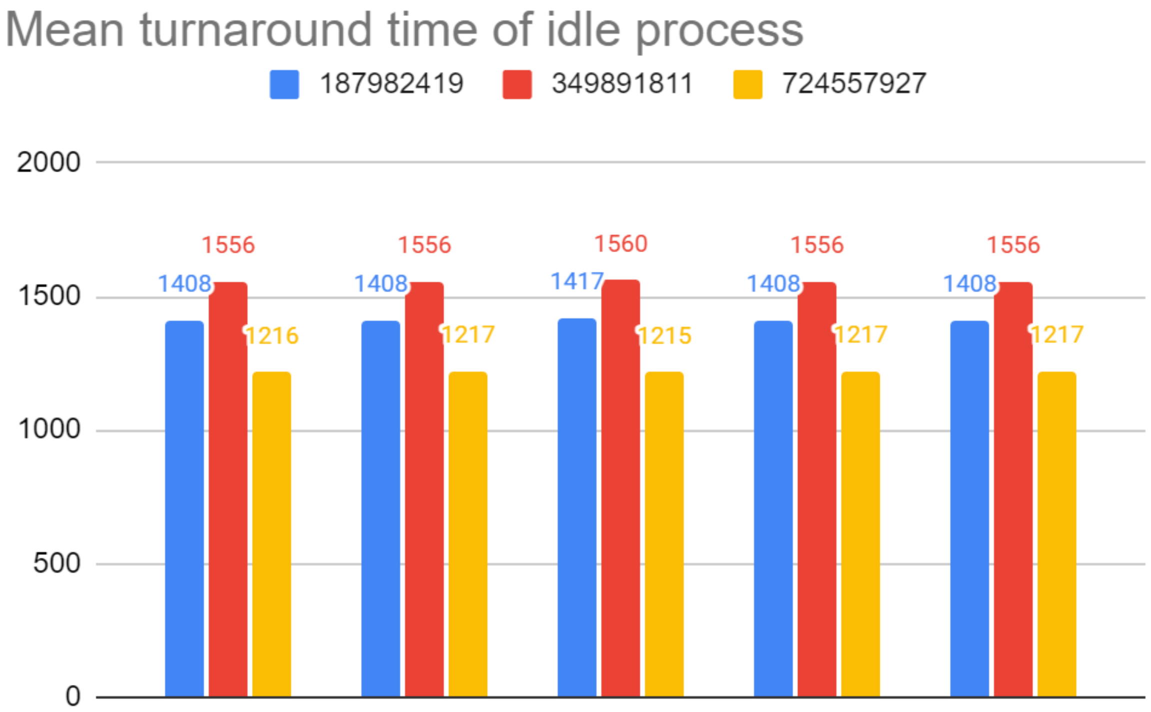
In this graph, we see very little difference in the turnaround time of the processes. This is surprising because as I said in my hypothesis I believed that the schedulers that could more effectively move processes on and off the core would have a lower turnaround time. However, despite this data being similar across the board, there is one outlier.

This anomalous scheduler is that of the “feedbackRRScheduler”. While not immediately noticeable, there is a clear decrease in turnaround time across all three of the seeds. On two of the seeds this is less apparent, only resulting in a decrease of about 1 time unit on seed “724557927” and an even smaller decrease of 0.5 on seed “349891811”. With seed “349891811” there is a much more apparent decrease of 3.2 time units. Where the others could simply be errors, I believe this third seed's input data reacted more favourably. Because of this decrease, it can be concluded that out of all of the schedulers, the data input of the three seeds worked better on the feedback round-robin. It also suggests that the input data itself can have a more profound effect on the time changes of the scheduler, or at least the feedback round-robin scheduler.

In figure 2 below, we can see the CPU time of the idle process.



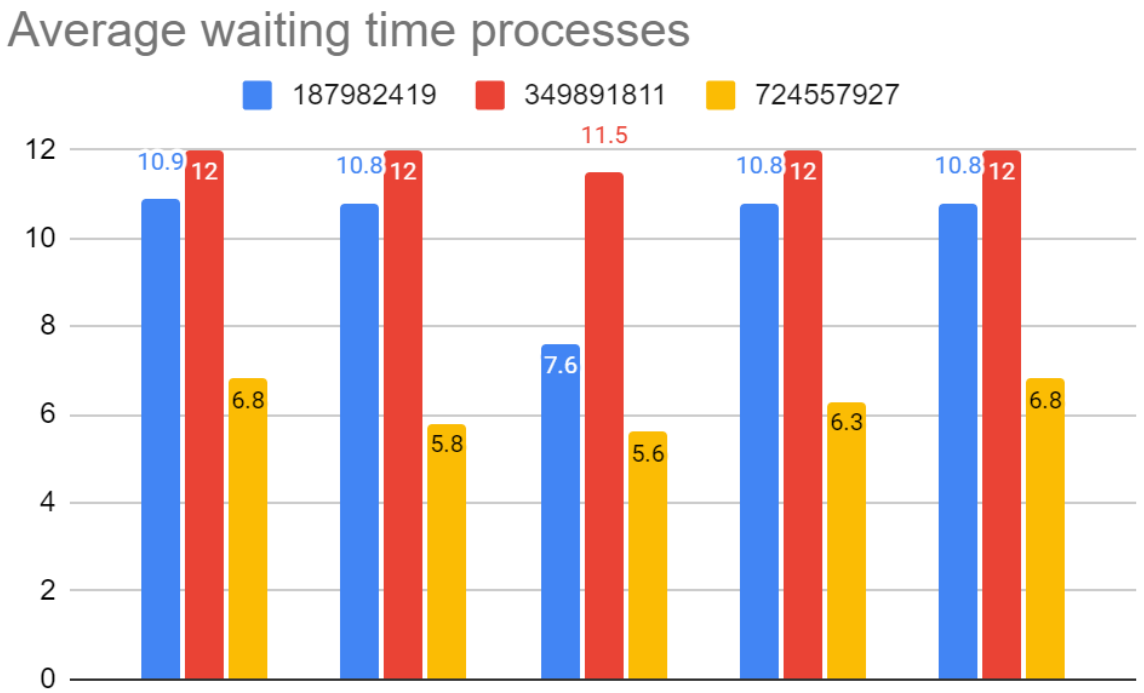
In figure 2, very similar to figure 1, there doesn't appear to be that much of a change between the schedulers. However, again there is a noticeable change seen in the feedback round-robin scheduler. Unlike figure 1 however, two of the three seeds cause an increase in the time of the idle process, which could mean that despite the fact that the processes have lower turnaround time, they are slightly less efficient as the CPU sits slightly more idle as it is moving processes on and off the core. This can be confirmed in figure 3 below, which shows the entire turnaround time of the experiment, which is correlated to the turnaround time of the idle process.



In the graph above, we can see a general increase in two of the three seeds for the feedback round-robin scheduler. As stated above, this means that though the turnaround time per process is decreased, the efficiency of the overall scheduler is decreased as the time taken for all of the processes to complete is increased. These changes are small, however, on a full-scale system with many hundreds of processes and many hundreds of bursts per process, the time changes will add up. This may result in increased waiting time for all processes to conclude, however, if you are only waiting on a specific process to conclude it will likely take less time to finish its work compared to other schedulers.

This is interesting, as you would assume that a decreased turnaround time for all processes would scale to time savings for the entire system. Though this would appear to be wrong, as these savings are surpassed by the fact that the system is taking longer to move things on and off the CPU, causing the idle time to increase.

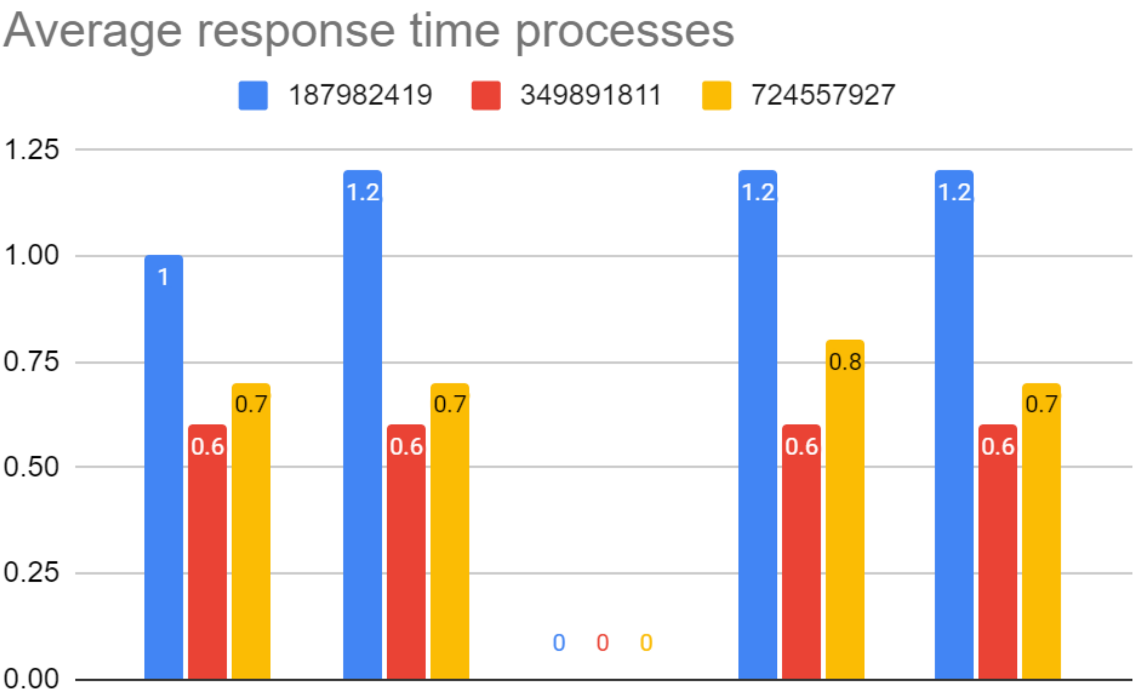
As seen with the other two figures, however, the time changes related to the other four schedulers are either minute or make no difference at all. This pattern changes slightly in figure 4, seen below, which represents the average waiting time of processes.



There is a clear difference in the mean waiting time of the processes on the feedback round-robin, this is usual as seen in the data shown. This decrease in waiting time is directly passed onto the overall turnaround time of the processes. This is corroborated by the fact that across all of the schedulers, the mean CPU time of the 10 processes is identical. This makes sense of course, as each of the schedulers has the same total amount of work to do, and time savings are more related to the efficiency of getting that work done. However, the important thing to note is the great difference in savings seen between the three seeds. This is a better representation of the statement seen above, below figure 1, which demonstrates the time savings compared to each of the other schedulers.

Despite this correlative data, which suggests little difference between the other four schedulers, in this graph, there is a clear difference in one of the seeds acting on two of the schedulers. These time savings in the waiting time, though minimal, are noteworthy because up to this point they have not been apparent. These time changes are demonstrated in figure 1, however, in figure 4 above, they are more clearly defined. This graph is very important, as it helps to demonstrate my hypothesis. With the mean burst time being so low in the standard RRscheduler, each burst is normally completed within its time quantum. Therefore, as it is non-preemptive on top of this information, it produces the same results as the first come first serve algorithm. This means, in a situation where this kind of data is created on processes, the savings made on the round-robin usually would not occur here but with the increased processing time on the scheduler's side. Therefore, if having to choose between these two schedules specifically, you would have a very slight edge over the regular round-robin when going with the very primitive and simple first come first served. This can be very slightly demonstrated here with the 0.1 decrease in waiting time on one of the seeds, though this is such a tiny difference seen here it could also just be a rounding error or general methodology error.

Finally, figure 5 demonstrates the average response time of the processes below.



As seen in the control data, there is a very similar trend in the proactiveness of the schedulers. This graph represents the time taken between the created and started time of processes, and like the control data, the noticeable scheduler is once again the feedback round-robin. I believe this is because of the fact that it is preemptive, and is the only of the five schedulers that actually acts with the override condition of preemptive. Without this activity, the behaviour is similar in regards to having the generally lower response time however not the extreme case as seen above, which is just 0.

There are slight changes, as seen in the seed “187982419” on the standard round-robin scheduler, and seed “724557927” for the SJFScheduler. Both of these cases could represent a slight approval of my hypothesis again, however, the fact that change is not represented in the standard FCFSScheduler does also balance this by providing evidence of it not making a difference.

Discussion -
As seen in the experimental data, there is only a noticeable difference for the one scheduler. This feedback Round Robin, which was the only scheduler to consistently show a change in the categories analysed. However, most of the changes seen in the data appear to corroborate my hypothesis. The data suggest that the standard, non-preemptive, round-robin algorithm behaves generally the same as the first come first served. As I suggested, I believe this falls down to the fact they are moving the same processes in the same order, with generally lower burst times than the time quantum. This is seen in most figures above, with only slight fluctuations of at most one-time units saved in figure 3 on only one of the seeds.
In my hypothesis, I suggested that the shortest job first algorithms will provide little benefit over that of the others, because of the case that all processes are similar in size and very short. This is seen in all figures above, with the exception of the ideal shortest job first seen in Figures 1 and 4. In figure 4 specifically, there is a slight decrease in the waiting time on two of the three seeds, one larger than the other. The fact that the changing seed causes different time differences suggests that it could provide a benefit in the overall behaviour of the CPU when given these seeds' specific data types.
Overall though, i didn't expect there to be such a noticeable difference in the feedback round-robin. This goes completely against my hypothesis, as I expected the feedback round-robin to not generally behave differently from its non-preemptive counterpart. Though its first level of priority has a lower time quantum of only 8 time units, I believed it would not generally be higher than this and would instead benefit in the same way of just getting tasks onto the CPU. This didn't happen in most cases, instead, it was more related to the input data that caused differences. Some of these differences were massive, such as the large change in the waiting time of seed “187982419” compared to the other two seeds. This is with exception of the response time in figure 5, which I didn't account for in my hypothesis at all.

Threats to validity -
I believe the data set I chose for this experiment was too small, and thus changes that appeared could have either been eliminated by more experimental data or exposed more greatly as well. The changes seen were generally so small, that they could just appear as errors, and with more processes, higher mean burst counts and more seeds, they could go either way to be greater things to make note of that weren't seen here or could be completely gotten rid of.
Furthermore, I believe that there was a bias towards the round-robin schedulers compared to the shortest job first ones. This is because the mean burst count was so small, the shortest job first scheduler didn't serve much of a purpose and in theory was likely to behave similarly to FCFS.
Another threat to validity is the behaviour of the response time. Once again, as seen in both experiment and the control data the response time on the feedback round-robin was very different to the other schedulers. This is down to the fact that it is preemptive, and it will interrupt new processes' arrival

Conclusions -
Generally, there are few differences between the schedulers, with the notable exception of feedback round-robin. This expectation comes with further specificity, however, as only specific seeds can truly benefit from this scheduler, whereas others have very little benefit. Overall, it appears that more data is needed to come to some proper conclusions, but seed data makes a difference in the behaviour of the schedulers. As well as this, round-robin schedulers with a larger time quantum (higher than the mean burst time) will also generally behave similarly to first-come first served algorithms.
Overall I expected more in regard to change, and I thought that by making many smaller processes you'd have a larger effect. I don't believe the data set I created was large enough, and this has resulted in data being very similar across all schedulers and all graphs, other than that of figure 5 which has created concern that the preemptive functionality of feedback round-robin is not working correctly.

Experiment 2 - An investigation into the efficiency of the scheduler when all processes arrive at the same time. This will also investigate the times related to the idle process.

Introduction -
This experiment will be an investigation into the turnaround time, waiting time and response time of many different processes when the created time is uniform across all processes. By having many different processes all arrive at the same time, but have differing quantities of CPU bursts and lengths, I will investigate the effect that the different scheduling algorithms have on the times of the processes, as well as look at the effect on the idle process.

I hope to demonstrate a difference in the idle process, as the CPU theoretically will not have anything to do when they all arrive at the same time. This will be demonstrated between the different schedulers to see how they take advantage of this fact and see which one is better for more continuous large ready queue tasks such as data analysis. Specifically, I hope to see a clear change in the CPU time and the turnaround time of the idle process, with an analysis of the turnaround average of each scheduler as well.

My hypothesis on the effect of the idle process will be a vast decrease in its operation time. This would mean that because all processes arrived simultaneously, the scheduler never needs to put the idle process onto the CPU as there is always something for the CPU to be doing, rather than waiting for things to arrive.

Methodology -

For the second experiment, a change to the input file will be made on the same three seeds as the control group. This means the data can be compared to a control group as well as each of the other schedulers. The change made to this input parameters file will be a change to the “meanInterArrival” subsection. This will be a change of setting the parameter to zero; This change will cause the arrival time of all of the processes to be the same 0 time of initialization. The other parameters will stay the same as the control group, this will consist of the experiment being run on 10 processes with an initial starting priority across the board of 0. Furthermore, the standard CPU and IO bursts will be set at 25, once again the same as the control group. This is because this is not an investigation into the burst times of the processes necessary, just the effect of efficiency on uniform arrival times before the bursts. Finally, the mean number of bursts will remain at 5, meaning the number of bursts each process has compared to the control will once again be the same.

In regards to the simulator parameters, once again no changes are made to the control group. This means a 10000 time limit to ensure that all processes are finished in their entirety. The period will remain false, and the interrupt time will remain as 0. The time quantum won't be changed either, as this is changed where needed in the feedback round-robin and will remain at 20 for comparable results with the control and other experiments when the time comes.

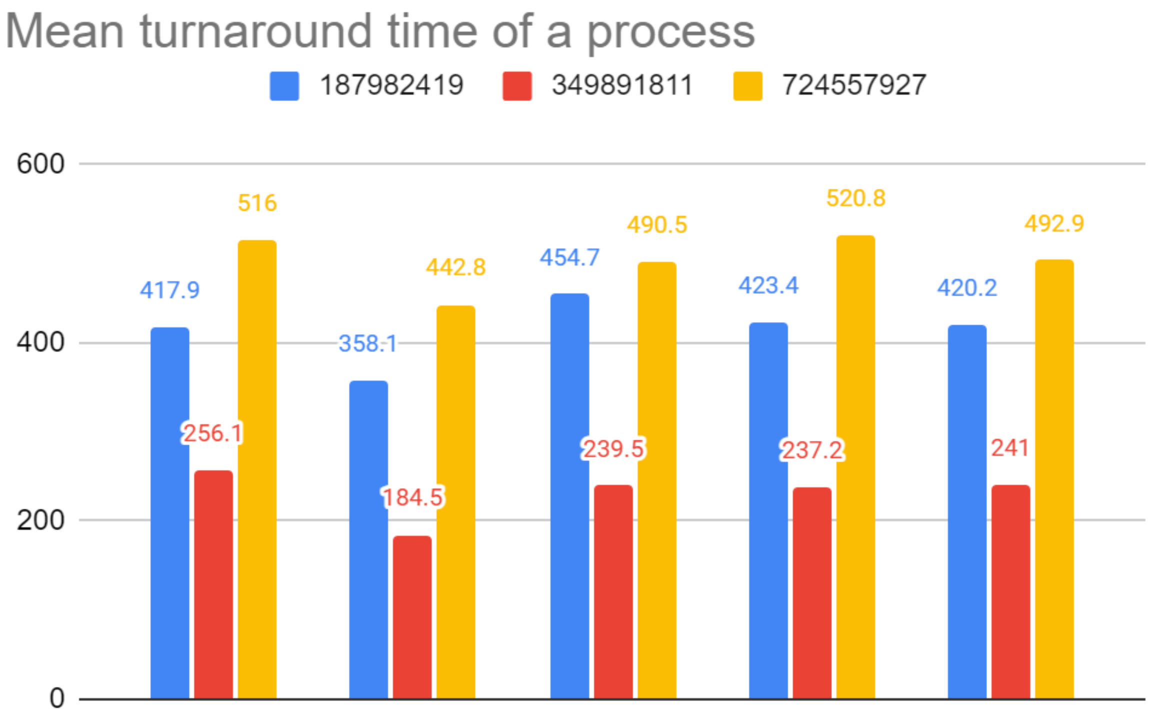
This experiment will remain very similar to the control in every way, but with this augmentation to the starting time, it will demonstrate the potential and drawbacks of each scheduler when coming across frequent process arrival. This will also act as a demonstration of how the schedulers benefit and suffer from large queues.

By comparing them to the control group, I can check for general trend changes. If there are clear errors in the data that affect all seeds and all schedulers, this can be compared further to the control to check for program-wide errors or it can be used in situations where a full renewal of data is required. As well as this, I have used three different seeds on the data, and the methodology of the experiment was applied to all of these. This is important, as having three data sets increases reliability, as one being wrong is very apparent. Three different seeds should be able to demonstrate clear similarities in the experimental data as well as show the drawbacks and inconsistencies of the methodologies used.

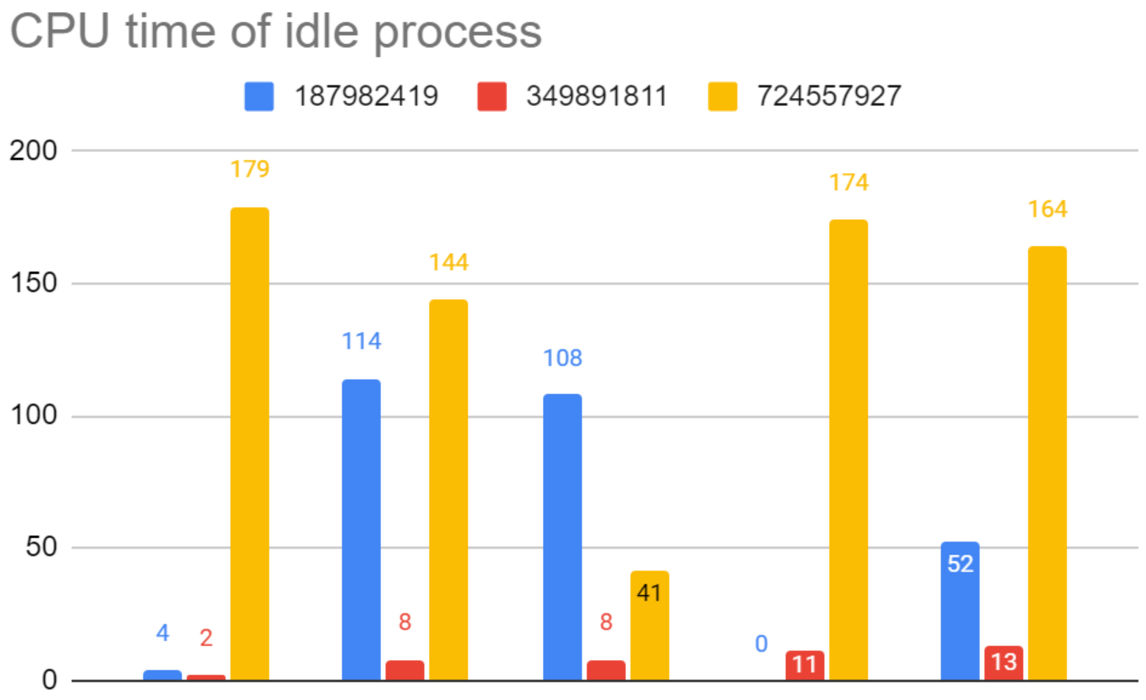
Results -
On all of the graphs shown, the data points from left to right are the five different schedulers. In order, they are;
RRScheduler, non-preemptive round-robin
IdealSJFScheduler, Shortest job first knowing next burst time
FeedbackRRScheduler, Preemptive feedback-driven round-robin
SJFScheduler, non-preemptive shortest job first using exponential averaging
FcfsSchdeduler, first come first served

In figure 6, the Mean turnaround time of a process is the mean value of the 10 processes on each of the schedulers. As shown in the chart, the data has one clear dip in turnaround time indicating a single scheduler of the five is more efficient generally. This, however, is dependent on the data set. As seen on the seed “187982419”, there is only a marginal increase in efficiency over any of the other schedulers on the same data set. This indicates that, though there is a clear decrease in time for some data sets, others appear to not benefit greatly from the ability to see the next process burst. This does not appear to be dependent on the input data either, as in the cases of the seeds used the input is spread more evenly between seeds "349891811” and “187982419”, compared to the chart which has a more similar result being shown between seed “349891811” and “724557927”. There is perhaps a slightly larger benefit to one process being larger than the others, however, it is not a clear increase in efficiency.

A clear wave pattern is starting to be established in this first graph, with round robin suffering for some reason with its quantum time interruption on two of the seeds in comparison to its FCFS counterpart. As well as this ideal SJF takes a clear lead in getting turnaround time down, with the benefit of less processing time to decide which process to put onto the core next as well as the benefit of knowing truly what the next shortest job is.

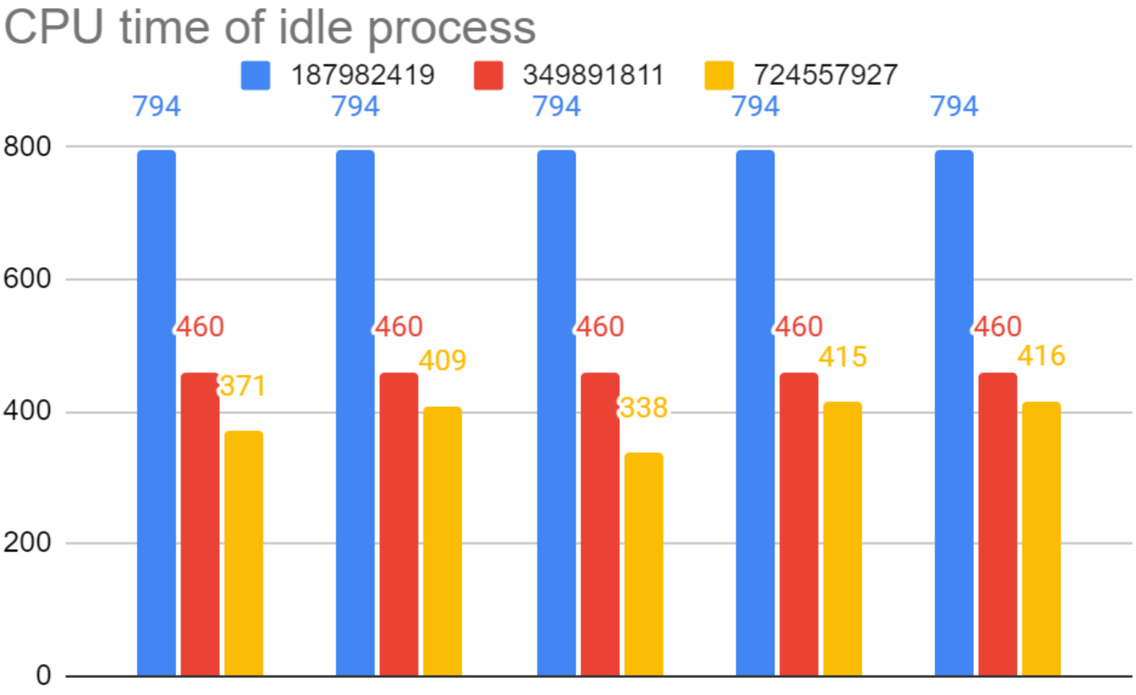


In figure 7, there is not much of note as there are no general trends represented in the data set I have. However, there is one piece of note and that is the shortest job first with exponential averaging scheduler acting on the seed “187982419”. This results in a CPU time of 0, indicating that using this method the idle process was not on the CPU at all. This is important as it shows high efficiency, as the CPU was never idle. I would like to note, that this specific set was repeated to ensure that the data was accurate using the methodology described above, and no changes occurred in the output sample. Furthermore, the blocked time of the same idle process also remained at 0, demonstrating that it was never used. Because of the lack of a trend, seen in figure 7 is the same data but on the original arrival time of the control data set.



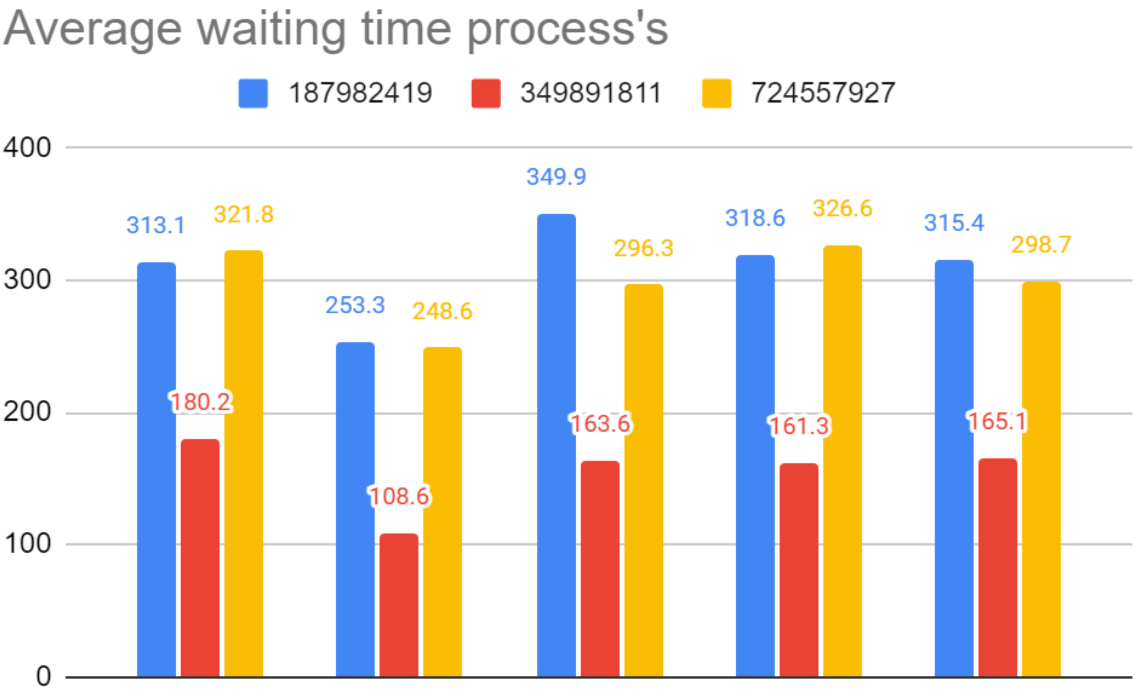
As seen when comparing the data sets, generally the scheduler has less effect on the CPU idle time than the arrival time of the data as well as input data spread. In figure 8, which is the control version of the same data set shown on the second page, there is no trend that can be related to figure 7 in the same way. All of the idle times of the processes are the same, with the notable exception of the feedback round-robin which on only one of the seeds is greatly reduced. There is a minor decrease in the same seed on the ideal shortest job first, however, it is such a small decrease it could be explained as an error.

Two of the three seeds show no movement whatsoever on any of the schedulers, which is a clear comparison to the ruminating seed which differs in 78 time units between peak and trough. This could be a source of data validity issues, though I think it is more likely to be an issue in how the different seeds don't have enough processes or number of bursts to truly show much data derivation.



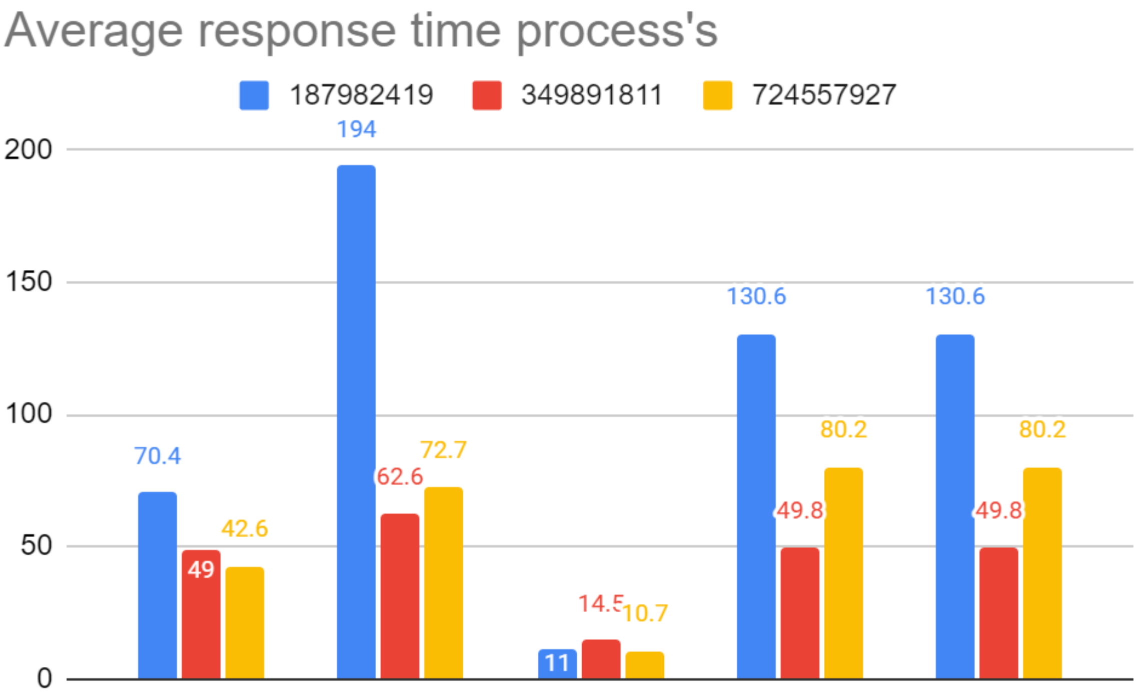
In figure 9, Average waiting time processes, we can see the mean waiting time of all processes on each seed and in each scheduler. The general trend here is different to that of figure 7, and more acts to demonstrate the better behaviour of the ideal SJF in regards to getting processes onto the core. It acts as the outlier in this way and helps to show the stark comparison between itself and the standard SJF algorithm.

Interestingly, the FCFS algorithm and RR both act very similarly on one seed, and FCFS acts similarly on the other two seeds. This demonstrates again the difference in input data causing the algorithms to suffer and thrive.

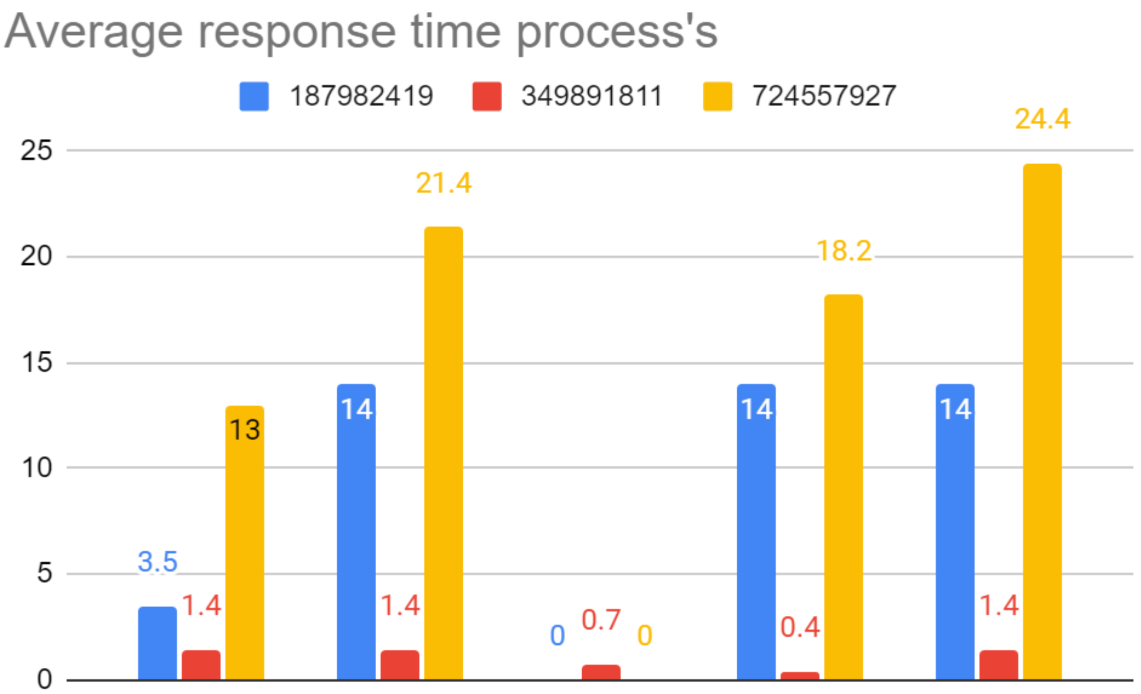


Finally, the analysis of the response time data. Clearly shown in this data is a vast decrease across all data types of the feedback round-robin. This can be put down to the fact that this scheduler is preemptive, meaning that the time between creation and being started is virtually the same, at least compared to all other schedulers.
Another immediate observation is the fact that in two of the three seeds, the response time for the ideal shortest job is larger than the others. This could be down to the fact that because it chooses the shortest job so accurately, it isn't directly responding unless the burst of that item is shorter than the other in the ready queue. This is in contrast to the feedback round-robin, which is likely to immediately choose the newest item as its priority is always 0 when it arrives. The others in the ready queue that it has already worked on however are most likely higher than 0, because they could have already been processed in some way and had their priorities raised making them less favourable to work on by the scheduler.

An interesting comparison to make is to look at the difference between the schedulers when acting semi-random arrival times compared to the uniform arrival time. This will be shown in figure 10. Seen in figure 10 is a very similar trend in the data, with a massive decrease in feedback round-robin thanks to its preemptive behaviour.



In figure 11 seen below, the data again seems to demonstrate the same behaviour as seen above. This is mainly the round robin demonstrating a very low response time, comparable to that of the feedback version. However, comparable is probably not the correct way as it is only similar when compared to the other data sets, and only in two of the seeds. The response time of the feedback round robin, like the previous experiment is virtually entirely 0. Once again, this is because of its preemptive behaviour however the standard round robin is not built preemptive in the eyes of the simulator. This is important, as it demonstrates a much better response time regardless of this, meaning it is likely that it is acting as first come first served with a timer that allows it to visit other processes just after creation.



Discussion - In general, I believe my hypothesis was correct. This is backed by a general decrease in CPU time for the idle process across all schedulers, some more so than others. This is demonstrated in figure 2 and figure 3. However, it is influenced by the data that is presented to it, not just the fact it doesn't have anything to do. This is seen in the difference between seed “724557927” compared to the other two. A larger data set could help prove this, however for now it is just one of three appearing to be different. Despite this, there is still a massive decrease, roughly half, for that seed still adding credibility to the hypothesis.

Threats to validity -
By having a static priority of 0, all the processes arriving at the same time have the same base priority. The result of this is that the feedback round robin is skipping straight to the newest item that is added to the queue, rather than staying with the current item. Even though it is behaving the best out of all the schedulers, by getting something onto core as quickly as possible it is not acting the best in terms of finishing jobs it has already started.

Another threat to validity is the behaviour of the SJF scheduler, with more testing a better alpha could've been chosen to make it benefit here. This didn't happen, and though it performed well in this test of extreme work the fact it was taking more time to decide what to do next rather than just getting on it meant it was making semi-poor decisions about what process would go next while having to spend time getting these decisions in the first place.

Conclusions -
Generally, the arrival time of the processes does appear to make a large difference across the board, and it can affect the efficiency of the schedulers independently of one another quite heavily. This is because of the fact that the schedulers are essentially swamped with

information, meaning that any scheduler that can benefit from short decision time and better-getting items on as quickly as possible will do better.
The shortest job first suffered here because of the lack of testing to get a better alpha. This is clear when compared to its ideal counterpart, which did much better in comparison. Therefore, if you took the benefits of the shortest job first in getting the quick processes out of the way and coupled it with the benefits of timer-based reactions you could have a better scheduler. Though this is a hypothetical better and it could also just couple the worst parts of both together.
Overall, this experiment demonstrated how the schedulers would act when having a backlog of data they need to handle. However, other than this because of the lack of priority information and testing to allow for better schedulers to be made, in the case of SJF, the experiment didn't yield the best results.

Experiment 3 - Reduced IO burst by a factor of 2 with an increased CPU burst also by a factor of 2. The mean number of bursts will remain the same, however, the meanInterArrival time of the processes will be decreased to 50, rather than 150.

Introduction -
This is intended to simulate a very CPU intensive set of processes that appear more frequently than the control but not all at the same time as in experiment 2. This will be an investigation into the effect on the different schedulers, of more CPU driven tasks. In this experiment, I hope to demonstrate how the different schedulers react when given more intensive tasks with more appearing more frequently.

Methodology -
The changes will be small but generally should have a larger effect. Increasing the CPU burst time, it means that the amount of time the scheduler is not doing something is increased. This small change is aimed to

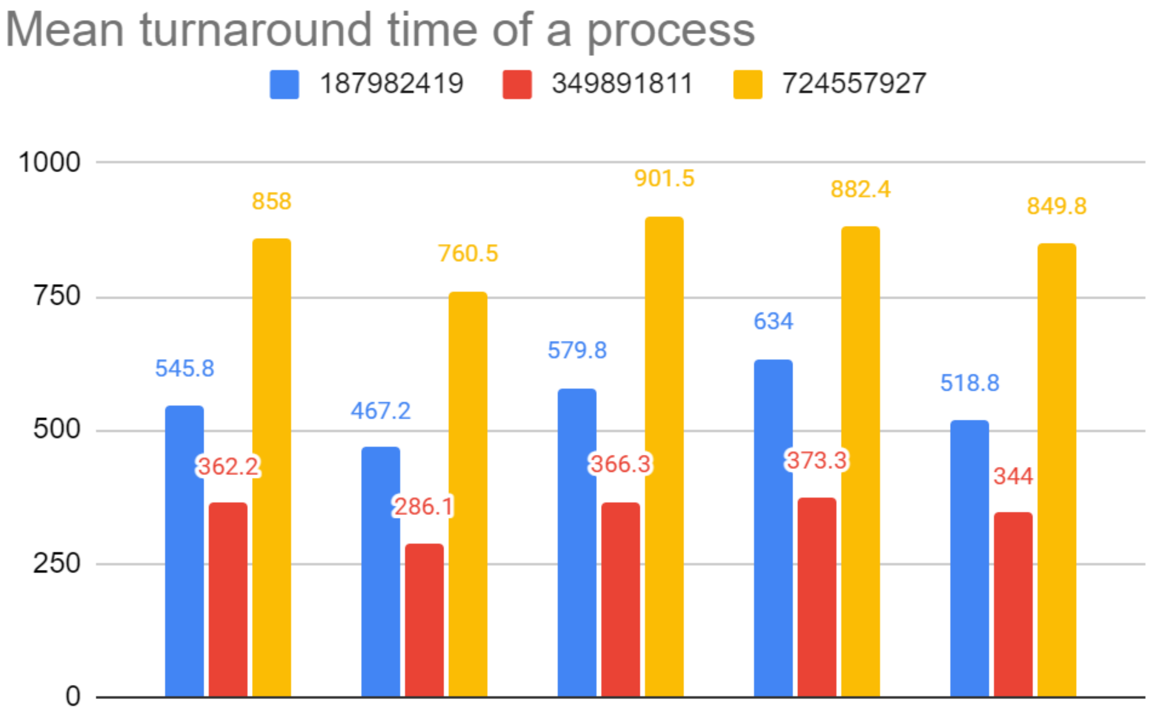
In regards to what's changing, the mean CPU burst is being raised by a factor of 2 from the control up to 50 time units mean, and the mean IO burst is being dropped by the same factor of 2. There is symmetry here, mainly to demonstrate a more CPU driven program being lined up over IO driven, as to take up more time actually on the core and try and raise waiting times. Furthermore, by having symmetry in the two io bursts, the overall time for processes to run to completion should be more similar to the control data.

However, this is not fully true as the mean arrival time will also be decreased. This means that the amount of work the scheduler has to do will be increased, as it tries to balance the more frequently arriving processes while the total amount of time the processes have to run roughly stays the same.

Hypothesis - I believe that in this experiment, the shortest job first algorithms may prove better compared to the round-robin schedulers. In the previous two experiments, round-robin has been better generally with its ability to act similarly to first come first served, as well as in the case of experiment 2 be more proficient in pure work by moving things as quickly as possible onto and off of the stack.

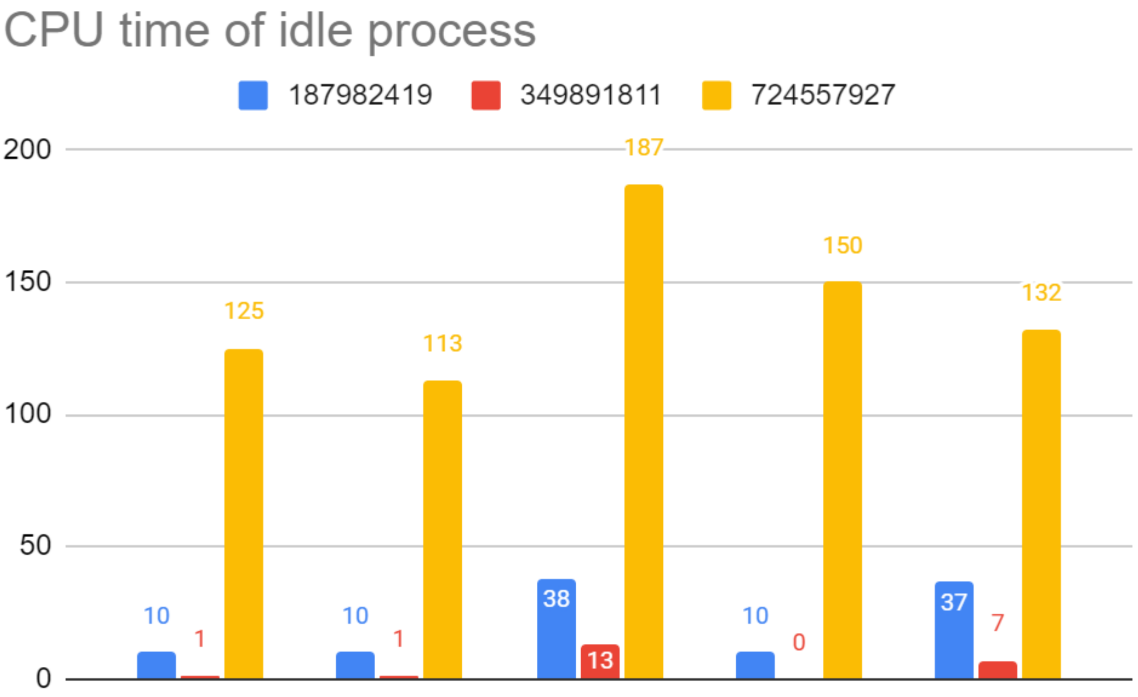
Following this, first come first served and the two round robins may suffer as more time is wasted. This could affect the feedback round-robin more so, as the lack of priorities here means that it will be wasting time analysing for the priority where it will not change, and its time will potentially add up. Furthermore, round-robin with its 20 quantum times will act more favourably, and likely more similar to the first come first served the same as experiment 1. Though the meantime is raised, it is still.

Results -
As seen below in figure 12, the Mean turnaround time of a process, there is hopefully a clear data pattern starting. Just for reference, the schedulers correspond to the data sets in the order; RRScheduler, IdealSJFScheduler, FeedbackRRSchdeduler, SJFScheduler, and FcfsScheduler.
It is apparent that the feedback round-robin and ideal shortest job first are the worst of the schedulers. This is like the previous experiments, dependent on the seeds. In the seed “724557927”, the feedback round-robin acts worse than the others, this is in comparison to the ideal shortest job first which is worse in the other two seeds. However, on the seed “349891811” all of the mean turnaround times are much closer to one another. This may be more related to the fact that the ratio is the same, however, specifically the difference feedback, ideal shortest and standard round-robin are much smaller for seed “187982419” than the seed “349891811”.



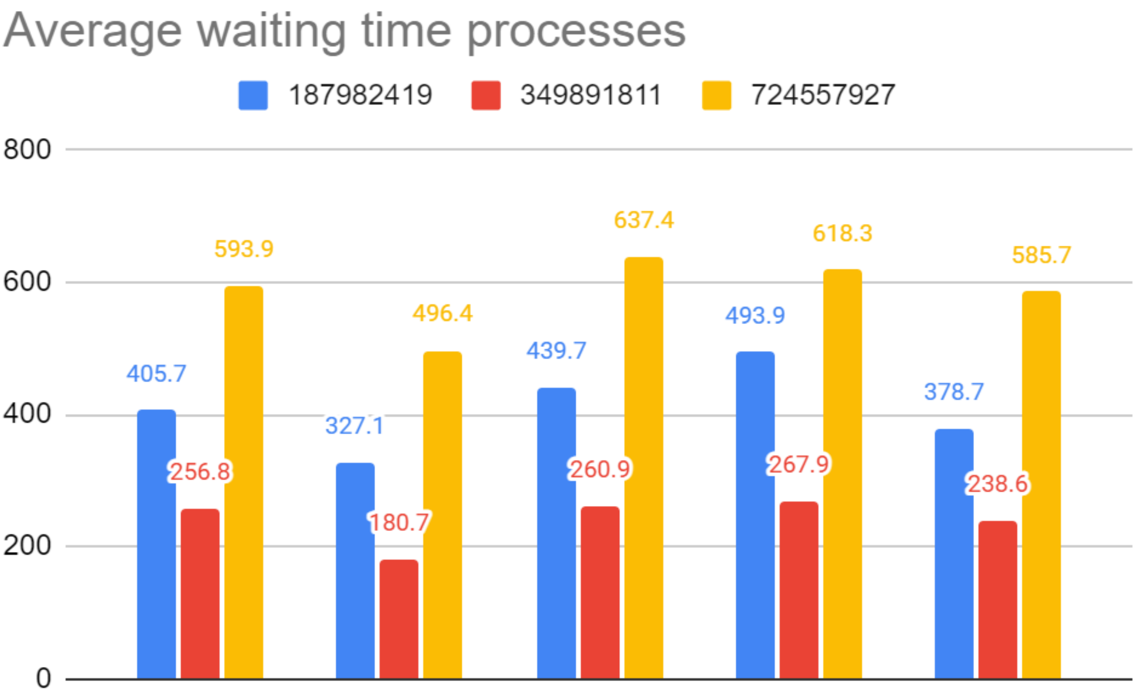
In figure 13, the CPU time of the idle process, a similar story is told as in figure 12. The feedback round-robin was the worst, with the highest amount of idle time in all three seeds. This suggests that the scheduler was spending more time taking items off and putting them onto the CPU. This can be presumed to be down to the fact that its standard quantum time was only 8 time units, compared to the now higher mean burst time of 50 this is going to mean it is much more likely to be taken off the core at least once, and likely twice.

As well as this, seen in both of the previous experiments so far, is the difference in the behaviour of the seeds. Both seeds “349891811” and “187982419” have got very similar and very low time units, with very little derivation. This could suggest that the seeds haven't had the potential time or the right combination of process and burst behaviours to allow for higher idle time, rather than there being something similar in the scheduler behaviours. This is countered by the fact that on the other two schedulers, the times are very different suggesting it is a combination of seed and scheduler behaviours. On the final seed, it can also be interpreted that potentially this is more scheduler related, as there has been more allowance for the idle time to build. Though, because the parameters are the same and caused the three seeds to produce very different results, this is again countered.



In figure 14, the trend continues once again with the ideal shortest job first taking the lead by a massive margin in its waiting time. This will directly correlate with the mean turnaround time shown in figure 12. This includes the trading of the highest waiting time between the two schedulers SJF and FeedbackRR over the three seeds back and forth.

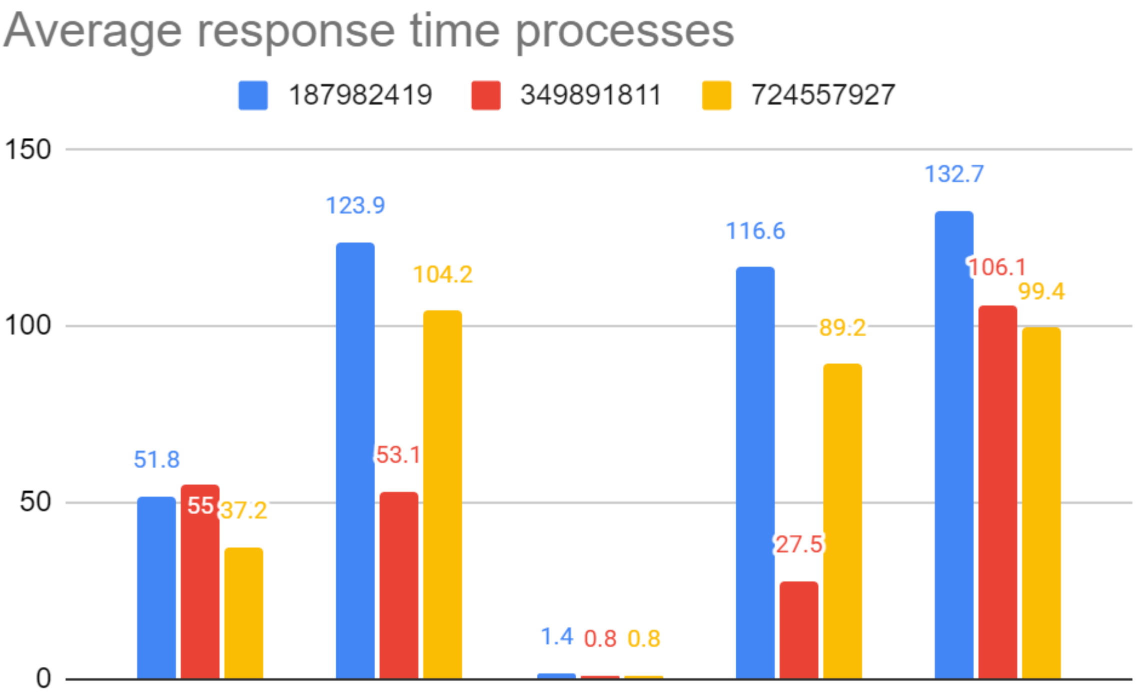
Something of importance is the similarity of the standard round-robin and first come first served. As I suggested in my hypothesis, these two algorithms behave in a similar manner across all three of the seeds, being within about 20 time units or so. Overall, these two are close in terms of waiting time similarly which is also directly correlated to them being close in mean turnaround time.



Seen below in figure 15, is the average response time of the processes. Something very important to note immediately is the fact that the round-robin scheduler, non-preemptive, acts very differently from the first come first served scheduler. It is understandable, that the first come first served schedulers response time would be high, this is because it has to fully go from front to back through the queue in order to respond, regardless of the time requirements. Compared to this, the round-robin scheduler does have a time quantum of only 20, meaning it acts in the same way as the first come first served but occasionally moves the first item to the back. This is presumably why its response time is half on one seed, and about a third on the other two.

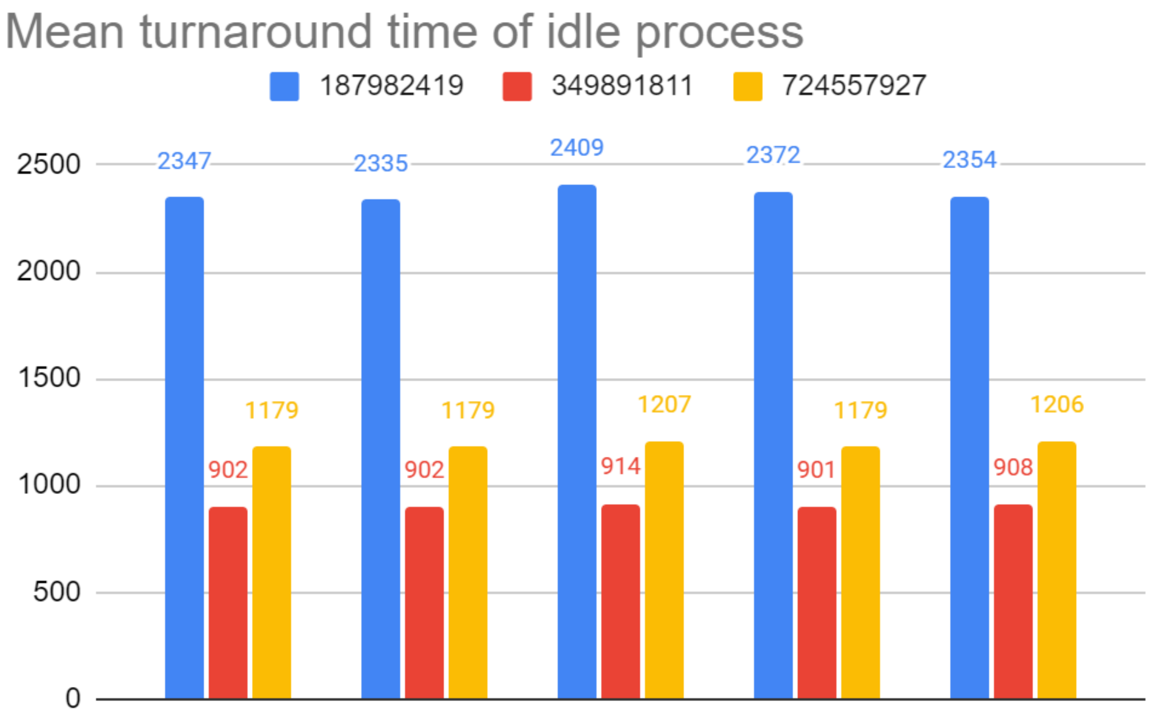
Another thing to note is once again, as seen in the other two experiments, the response time of the feedback round-robin is practically 0. In this experiment, this seems to make more sense, as it is a semi-random arrival of processes that are all likely lower priorities the response to these new processes should be immediate. This is because the scheduler is preemptive, and the moment the new processes arrive it will stop what it is doing immediately to respond to this new arrival.

The trend is not continued as seen in the other figures above with regards to the shortest job first scheduling. This is because they are non-preemptive, resulting in more of a chance that new processes are selected rather than a guarantee which would lower the time. As well as this, I surmise that the fact that the ideal shortest job first has a slightly higher response time because it doesn't make mistakes, unlike its more guess-related counterpart.



Finally, in figure 16 seen below, there is the mean turnaround time of the idle process. In figure 16 there is a clear trend in the data, similar to that of Figures 12 and 13. Specifically, in figure 13 there is a clear similarity between the seeds “187982419” and “349891811” on schedulers RR, Ideal SJF and SJF. They are related to the fact that the turnaround time of the idle process is directly related to CPU time, but it is an important representation of how close these schedulers behave under certain seeds.

A strange thing of note is the similarity between both the shortest job first algorithms and the standard round-robin algorithm, in regards to the idle turnaround time. This is because, in previous data, like the average response time where the schedulers have to put the idle process on while processing the next item, the times are much lower in the round-robin.



Discussion - My hypothesis stated that I thought that the round-robin and first come first served schedulers would act similarly, on the basis that they normally act similarly as the burst time is usually lower than the quantum time I have provided. This turned out to be fairly true, at least in the actual execution of the processes and handling. In regards to the idle process, the round-robin scheduler fell more in line with the two versions of the shortest job first schedulers. This is mainly true on the second two seeds however it is still very clear from the identical times on these two and the very close times on the remaining one.

Secondly, I suggested that overall the shortest job first algorithms would benefit better than the round-robin and FCFS algorithms, on the basis that the burst times being higher would mean the round-robin schedulers specifically would spend more time moving on the core. This is partially true, as the waiting time and turnaround time of processes was clearly much higher for the feedback round-robin compared to the ideal shortest job first. But this is not the full story, as seen in figures 12, 13 and 14 the SJF scheduler and feedback round-robin were comparable to one another, with SJF even being worse on two seeds in waiting and resultantly overall turnaround time. I had not predicted this, as I would have thought that even though the SJF wouldn't work as efficiently as its ideal counterpart, it will still function better considering it ran bursts to completion regardless of length. This is peculiar behaviour, as it means that the formula that was made to predict the next shortest job did not do a good job. Ideal shortest job first here shone out among the rest as it could truly get the turnaround times down on the processes that needed it, whereas the calculated predictions, essential assumptions and blind guessing, that the SJF made it act much worse. This will be raised as a threat to validity, as choosing an alpha of 0.5 rather than probing for the best one depends on the input data provided and making a more informed decision on it could make it behave much closer to the ideal version.

Finally, in figure 15 there is an unexpected change in behaviour regarding the round-robin scheduler. Even though in the other figures the ideal shortest job first algorithm behaved more effectively in ensuring processes had a quick turnaround and lower idle time on the core, the round-robin acted better in ensuring that the response time of all processes was shorter. This is especially interesting considering the other round-robin acted preemptively and got almost 0 response time and this one was not preemptive. As stated above, I believe this is because it acted in a similar nature to round-robin with interrupts, causing newly arrived processes to get responded to in some capacity better than FCFS. Therefore, if there was a way to combine the behaviour of the shortest job first, with adjustments to make it more similar to the ideal version, with the interrupting nature of RR, a better alternative could be made for these more CPU-intensive processes.

Threats to validity -
Similarly to the previous experiments, I believe the data set I chose was too small. This is in regard to the number of processes as well as the number of seeds I chose. The three seeds I chose behaved very differently, across all experiments and across all schedulers. When looking at the input data the seeds “349891811” and “187982419” were more closely aligned, at least in regards to the length of processes and spread of workload. This is a stark contrast to the behaviour demonstrated in the graphs above, with the seeds “349891811” and “724557927” being much closer in results. Therefore, I believe that there is more to it than its initial appearance and more seeds would have resulted in a better spread of data. As well as this, more processes would have also helped to bolster the data set and provide a better mean by where it was acquired.

Another threat to validity is the fact that the priorities of the data were all exactly the same. This means when it came time for the feedback round-robin to work, whenever a new process arrived it would more than likely be immediately put onto the core rather than letting the other processes finish. This was meant to be more similar to a general workload of a computer, with small IO bursts and large CPU bursts, however, if that was the case each process arriving would have a different priority. Because the CPU burst was so much higher than the chosen quantum times of the feedback scheduler, it more frequently triggered the full quantum time used and raised priorities which thus allowed the new processes to take first place. If this was changed slightly, and the static priority put to either 3 or 4, this scheduler would have acted better and basically produced the same results as first come first served with preemptive interrupts that stirred the queue.

By not working on the formula that the SJF scheduler uses, it also means that it created worse data than it could have done. By using previous data, a better alpha could have been chosen that adapts depending on the input data that it is being provided with. This could change alongside the values themselves, by having a for loop run alongside this it could check the previous burst times and predicted burst times by adapting alpha values to get the most accurate burst time. From this, the new alpha could be used for a few bursts and checked again to see how it is doing, and updated again if need be. This would mean for some data the scheduler could work more efficiently, and it would be more adaptable to different data types. Though this is counteracted by the extra time required every x bursts to check and then potentially recalculate the alpha value.

Conclusions -

This experiment showed that the round-robin scheduler acted in the same manner as that of the FCFS scheduler, however with the added benefit of the timer causing the renewal of a process. This meant that every once in a while when the time quantum was fully used, the next item in the queue was moved to the front allowing for a new process to get its time on the core. This meant that the response time was lowered, as well as helping generally with the idle process. The clear benefactor of this data set however was the ideal SJF, which proved the best overall of the five. This also contrasts nicely, as seen in all figures above, with the real SJF which estimated the next one.

Both the SJF and the feedback round robin suffered greatly during this experiment, showing how the shorter interrupt times of feedback RR caused issues in comparison to its 20 time quantum RR counterpart, as well as how the standard SJF algorithm suffered next to its perfect estimating ideal counterpart.

As well as this, like the other two experiments it is clear that I didn't create a large enough data set be it in either process or in the number of seeds. Input data created a vast spectrum over the seeds, however as demonstrated in the previous experiments it once again caused issues. This is seen clearly in figure 16 which only really showed variation across the schedulers on one of the three seeds. This is both a threat to validity and a reason why this data shouldn't be taken at face value.

Finally, I would like to say that In this experiment, unlike the previous two in my opinion, the data was more varied and could be interpreted in different ways. This to me suggested that it was a better set of parameters to change, as it presented weaknesses in the algorithms I designed to work as the schedulers. Overall, the different schedulers did well in demonstrating the different ways they run in this data set. The shortcomings of the schedulers are also shown very well, as this is a more accurate representation of the data types that the schedulers would actually be put through.